Towards Hard Real–Time Erlang

Vincenzo Nicosia [1]

[1] Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania – Italy

Sixth ACM SIGPLAN Erlang Workshop ICFP 2007
5 Oct. 2007 – Frieburg

Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

Functional programming for HRT: Erlang ?

- Functional programming paradigm can help a lot in modeling, defining, developing, testing and maintaining RT systems
- In particular, Erlang/OTP has been successfully used for massively concurrent soft real–time systems
- Erlang/OTP gives some basic functionalities that are really useful in developing RT systems:
  - A huge and complete standard library
  - OTP, which gives a lot of power and flexibility to manage large systems with a lot of cooperating processes even in distributed environments
  - The possibility of building and deploy embedded Erlang applications in an easy and reliable way
- We think that Erlang has much to say even in the field of HRT systems, but it lacks native HRT support!

Functional programming for HRT: Erlang ?

- Functional programming paradigm can help a lot in modeling, defining, developing, testing and maintaining RT systems
- In particular, Erlang/OTP has been successfully used for massively concurrent soft real–time systems
- Erlang/OTP gives some basic functionalities that are really useful in developing RT systems:
  - A huge and complete standard library
  - OTP, which gives a lot of power and flexibility to manage large systems with a lot of cooperating processes even in distributed environments
  - The possibility of building and deploy embedded Erlang applications in an easy and reliable way
- We think that Erlang has much to say even in the field of HRT systems, but it lacks native HRT support!

Functional programming for HRT: Erlang ?

- Functional programming paradigm can help a lot in modeling, defining, developing, testing and maintaining RT systems
- In particular, Erlang/OTP has been successfully used for massively concurrent soft real–time systems
- Erlang/OTP gives some basic functionalities that are really useful in developing RT systems:
  - A huge and complete standard library
  - OTP, which gives a lot of power and flexibility to manage large systems with a lot of cooperating processes even in distributed environments
  - The possibility of building and deploy embedded Erlang applications in an easy and reliable way
- We think that Erlang has much to say even in the field of HRT systems, but it lacks native HRT support!

Functional programming for HRT: Erlang ?

- Functional programming paradigm can help a lot in modeling, defining, developing, testing and maintaining RT systems
- In particular, Erlang/OTP has been successfully used for massively concurrent soft real–time systems
- Erlang/OTP gives some basic functionalities that are really useful in developing RT systems:
  - A huge and complete standard library
  - OTP, which gives a lot of power and flexibility to manage large systems with a lot of cooperating processes even in distributed environments
  - The possibility of building and deploy embedded Erlang applications in an easy and reliable way
- We think that Erlang has much to say even in the field of HRT systems, but it lacks native HRT support!

Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

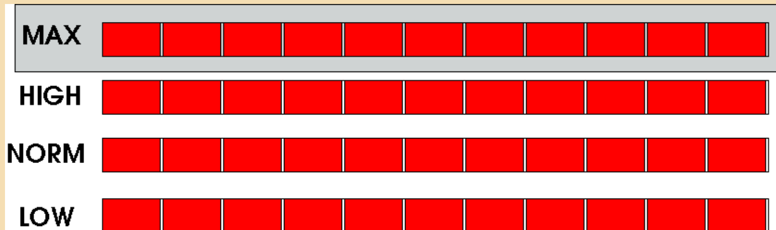
The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

Scheduler Queues



Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

Towards RT Erlang processes

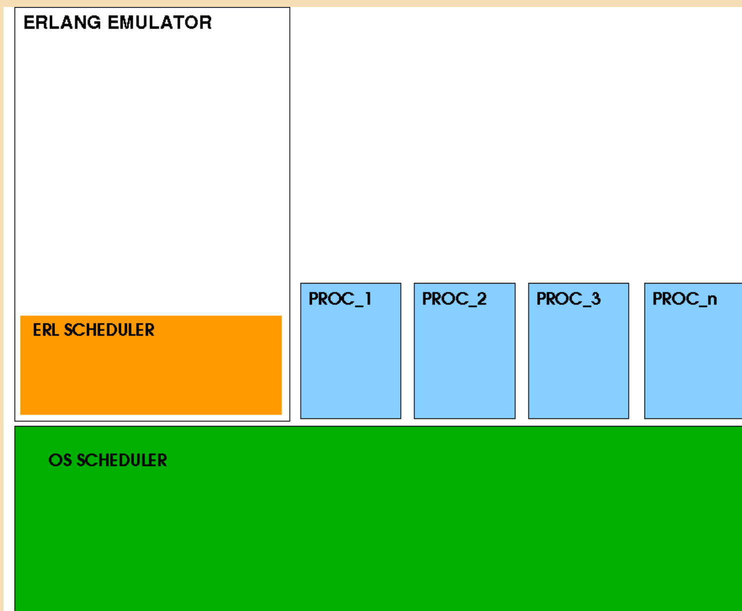
In order to have HRT capabilities in Erlang, three different approaches are possible:

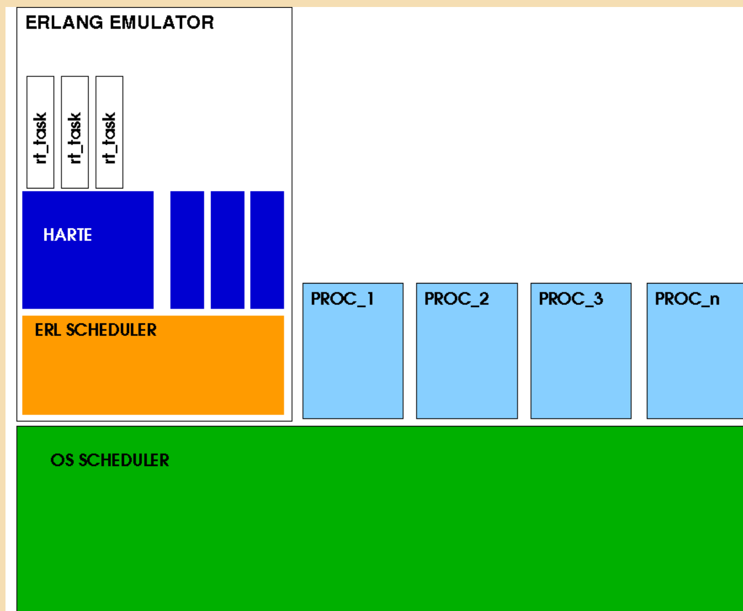
- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)

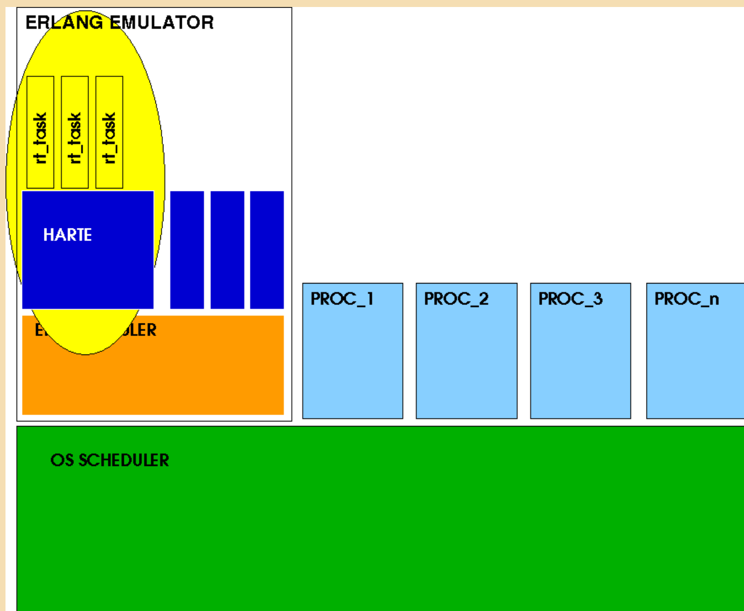
Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

ERLANG EMULATOR
PROC_1
PROC_2
PROC_3
PROC_n
OS SCHEDULER


ERLANG EMULATOR
ERL SCHEDULER
PROC_1
PROC_2
PROC_3
PROC_n
OS SCHEDULER


ERLANG EMULATOR
rt_task
rt_task
rt_task
HARTE
ERL SCHEDULER
PROC_1
PROC_2
PROC_3
PROC_n
OS SCHEDULER


ERLANG EMULATOR
rt_task
rt_task
rt_task
HARTE
PROC_1
PROC_2
PROC_3
PROC_n
OS SCHEDULER

HARTE: an overview

- HARTE is basically an application (a peculiar one), which is in charge of scheduling RT task
- In order to guarantee RT scheduling of task, HARTE itself runs with MAX priority (*)
- All HARTE tasks (i.e. HRT tasks the user would run), are created as low priority tasks and put in a scheduling queue using a Deadline Monotonic (DM) scheduling algorithm
- Then the scheduler is started and it schedules tasks one by one, modifying the priority of the task to be run to HIGH (*)

HARTE: an overview

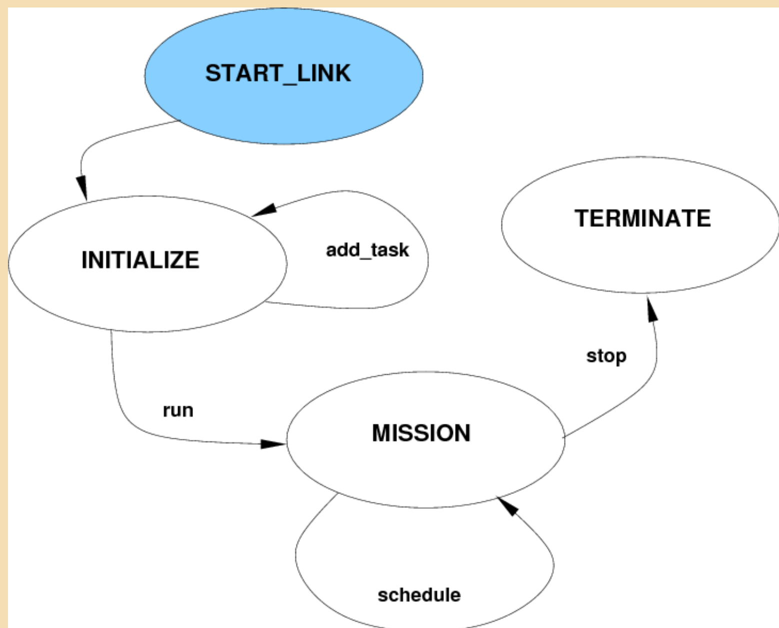
- HARTE is basically an application (a peculiar one), which is in charge of scheduling RT task
- In order to guarantee RT scheduling of task, HARTE itself runs with MAX priority (*)
- All HARTE tasks (i.e. HRT tasks the user would run), are created as low priority tasks and put in a scheduling queue using a Deadline Monotonic (DM) scheduling algorithm
- Then the scheduler is started and it schedules tasks one by one, modifying the priority of the task to be run to HIGH (*)

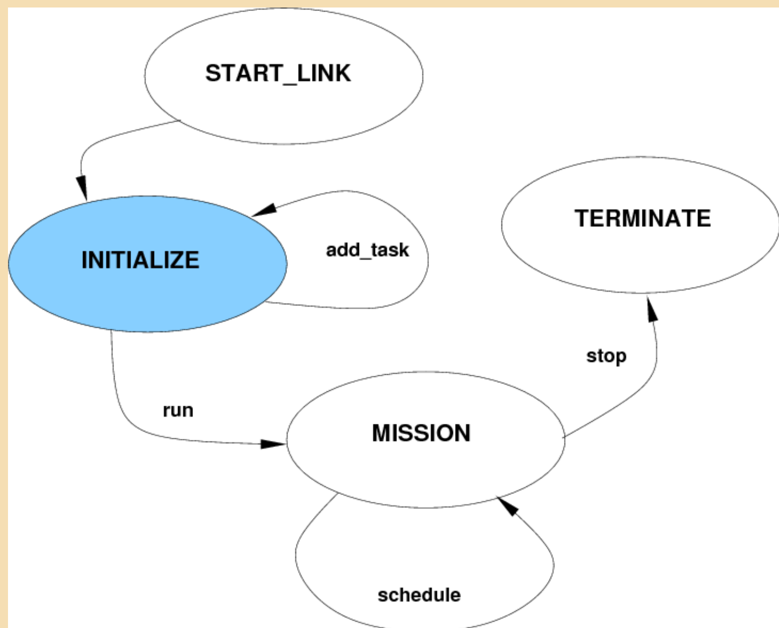
HARTE: an overview

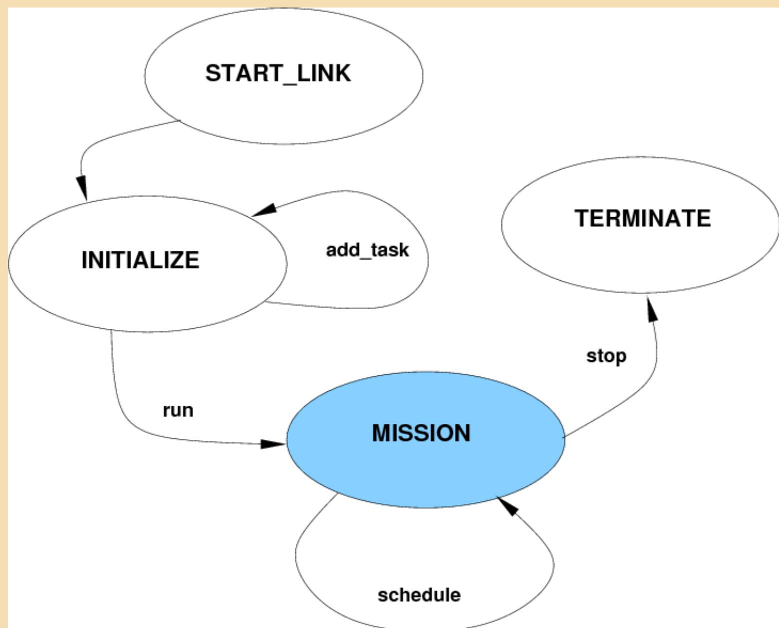
- HARTE is basically an application (a peculiar one), which is in charge of scheduling RT task
- In order to guarantee RT scheduling of task, HARTE itself runs with MAX priority (*)
- All HARTE tasks (i.e. HRT tasks the user would run), are created as low priority tasks and put in a scheduling queue using a Deadline Monotonic (DM) scheduling algorithm
- Then the scheduler is started and it schedules tasks one by one, modifying the priority of the task to be run to HIGH (*)

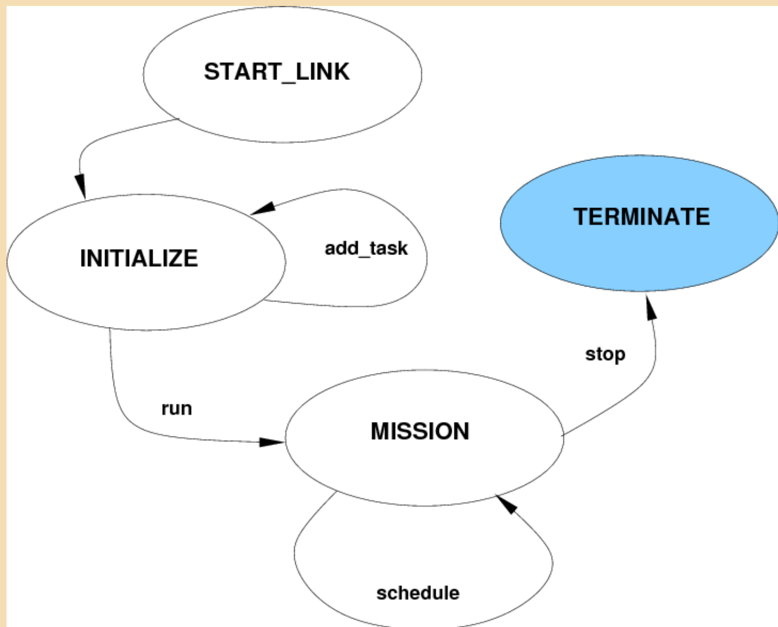
HARTE: an overview

- HARTE is basically an application (a peculiar one), which is in charge of scheduling RT task
- In order to guarantee RT scheduling of task, HARTE itself runs with MAX priority (*)
- All HARTE tasks (i.e. HRT tasks the user would run), are created as low priority tasks and put in a scheduling queue using a Deadline Monotonic (DM) scheduling algorithm
- Then the scheduler is started and it schedules tasks one by one, modifying the priority of the task to be run to HIGH (*)

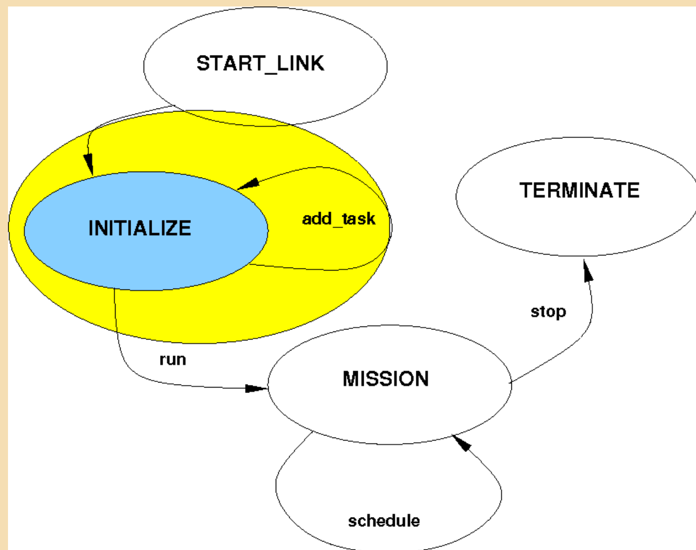

START_LINK
INITIALIZE
add_task
TERMINATE
stop
run
MISSION
schedule


START_LINK
INITIALIZE
add_task
TERMINATE
stop
run
MISSION
schedule


START_LINK
INITIALIZE
add_task
TERMINATE
stop
run
MISSION
schedule


START_LINK
INITIALIZE
add_task
TERMINATE
stop
run
MISSION
schedule

Details: init



Initialisation

- A new behaviour called rt_fsm has been defined. It is basically a gen_fsm with some additional code for RT management
- Each HRT task is represented by an rt_fsm
- In the initialisation phase, all RT tasks are defined and added to the scheduler
- To add a task to the scheduler, the init function of rt_fsm calls rt_scheduler:add_task(),

Initialisation

- A new behaviour called rt_fsm has been defined. It is basically a gen_fsm with some additional code for RT management
- Each HRT task is represented by an rt_fsm
- In the initialisation phase, all RT tasks are defined and added to the scheduler
- To add a task to the scheduler, the init function of rt_fsm calls rt_scheduler:add_task(),

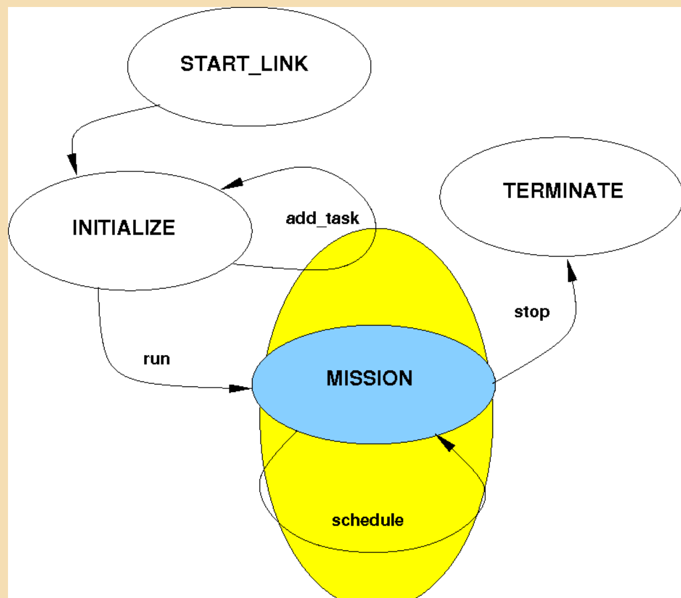
Initialisation

- A new behaviour called rt_fsm has been defined. It is basically a gen_fsm with some additional code for RT management
- Each HRT task is represented by an rt_fsm
- In the initialisation phase, all RT tasks are defined and added to the scheduler
- To add a task to the scheduler, the init function of rt_fsm calls rt_scheduler:add_task(),

Initialisation

- A new behaviour called rt_fsm has been defined. It is basically a gen_fsm with some additional code for RT management
- Each HRT task is represented by an rt_fsm
- In the initialisation phase, all RT tasks are defined and added to the scheduler
- To add a task to the scheduler, the init function of rt_fsm calls rt_scheduler:add_task(),

Details: Mission



Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused......or even if used... :-)

Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

Tests

- HARTE is still a proof–of–concept but it seems to work!
- We tested it with a couple of heavy CPU–bound benchmark, running both HARTE tasks and normal erlang processes at the same time
- The scheduler overhead, in different configurations, is reported in table:

Test	**Average Overhead (μs)**	**Std. dev.**
RT tasks only	13.1	245.78
RT and Non–RT tasks	11.7	52.5
RT tasks many periods	17.9	132.05

Figure: Average and standard deviation of scheduler overheads

Tests

- HARTE is still a proof–of–concept but it seems to work!
- We tested it with a couple of heavy CPU–bound benchmark, running both HARTE tasks and normal erlang processes at the same time
- The scheduler overhead, in different configurations, is reported in table:

Test	Average Overhead (μs)	Std. dev.
RT tasks only	13.1	245.78
RT and Non–RT tasks	11.7	52.5
RT tasks many periods	17.9	132.05

Figure: Average and standard deviation of scheduler overheads

Tests

- HARTE is still a proof–of–concept but it seems to work!
- We tested it with a couple of heavy CPU–bound benchmark, running both HARTE tasks and normal erlang processes at the same time
- The scheduler overhead, in different configurations, is reported in table:

Test	Average Overhead (μs)	Std. dev.
RT tasks only	13.1	245.78
RT and Non–RT tasks	11.7	52.5
RT tasks many periods	17.9	132.05

Figure: Average and standard deviation of scheduler overheads

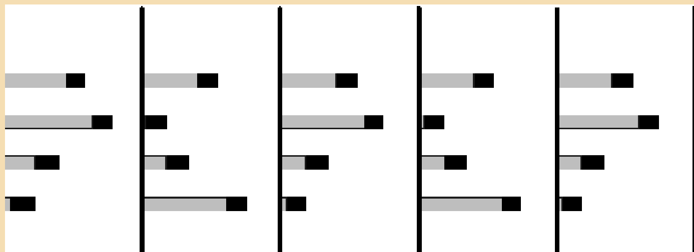
Tests

- HARTE is still a proof–of–concept but it seems to work!
- We tested it with a couple of heavy CPU–bound benchmark, running both HARTE tasks and normal erlang processes at the same time
- The scheduler overhead, in different configurations, is reported in table:

Test	Average Overhead (μs)	Std. dev.
RT tasks only	13.1	245.78
RT and Non–RT tasks	11.7	52.5
RT tasks many periods	17.9	132.05

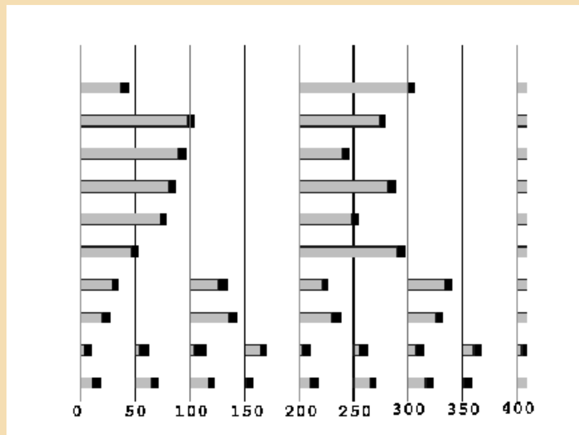
Figure: Average and standard deviation of scheduler overheads

Test 1: four HRT tasks



Test 2: RT tasks and 50 normal erlang processes

Test 3: HRT tasks with different periods



Outline

1. Motivations
2. Scheduling in Erlang
3. HARTE: A proposal for RT Erlang
4. Tests
5. Open Issues

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

"That's all Folks!"